Systems programming

7 – Sockets

MEEC LEEC MEAer LEAer MEIC-A

João Nuno Silva

TÉCNICO LISBOA

Bibliography

- Beej's Guide to Unix IPC
  - Chapter 11
- The Linux programming interface
  - Chapter 56, 57, 57
- Solaris SUN Programming Interfaces Guide
  - Ch 6, 7

Pipes / Names Pipes

- Only local Communication
- Impossible “private” communication
- Impossible one to one
- Impossible directed
- Only one protocol implemented
  - Stream communication
  - No messages boundaries

Sockets

- Transparency
  - Communication inter/intra machines is the same
- Compatibility
  - With existing communication mechanism
-
- Stream oriented
- Message Oriented
- Use of File system mechanisms

Sockets

- Introduced in 1981 on BSD 4.1
- It is an API
  - that define access points to applications
    - following the client-server architecture
- Sockets programming
  - more complex than files
  - More parameters
  - More system calls
- Main difference between FS base and socket based communication
  - How channels are opened and created.

Sockets

- Transparency
- Compatibility
    - Use of files
        - to reference channels
    - Use of the Regular I/O API
        - Send/receive data

Sockets types

Message oriented Sockets

- Each write is a message
- No message interleaving in the channel
- Each message is read atomically
- No data interleaving
  - Concurrent writes to not affect each other
- A read is concluded only after the conclusion of the write
- Two writes imply two reads

Message oriented sockets

Definition of a communication point	socket	Telephone
Assignment of a address to a communication point	Bind /address	Assignment of phone number
Send message	sendt()	Send SMS
Receive message	recvfrom(	Receive SMS
End of communication	close()	Turn off phone

Stream oriented Sockets

- Multiple writes form a stream/flow of data
- No data interleaving in the channel
- Each piece of data is processed atomically
- No data interleaving
  - Concurrent writes to not affect each other (system serializes them)
- Reads are serialized
  - 2nd read starts after the first

Stream oriented sockets

Definition of a communication point	socket	Telephone
Assignment of a address to a communication point	Bind /address	Assignment of phone number
Listen to incoming connections	listen	connection of a phone to the network
Start connection	connect()	phone call initiator dials destination number
Receiver established connection	accept()	receiver accepts call lifting the handset
Send / receive of data	send(),recv(	talk
End of communication	close()	lowering of handset

Message Reception

- On other IPC how receives messages?
  - Any process that open the channel
- Does the sender know the identity?
  - No
- How to solve
  - Assign each channel an address
  - Only one process can read “from” one address

Socket Types

- The socket type determines
  - The characteristics of communication
    - Delivery guarantees, ordering guarantees, communication directions
- Are defined as constants staring with the SOCK_ prefix
- **SOCK_STREAM**
  - stream oriented socket / connection oriented
- **SOCK_DGRAM**
  - Message oriented sockets / datagram socket / connectionless

Common socket types

- SOCK_DGRAM
  - Non reliable delivery
    - Packets can be lost or changed
  - No order guarantee
  - Non existing connection
    - Application should define recipient address for each message
  - (Uni/Bi)directional
    - Recipient can retrieve sender address and reply

- SOCK_STREAM
  - Reliable delivery
  - Ordered delivery
    - 1st packet to be sent is the 1st to be received
  - Connection oriented
    - Connection setup required before sending messages
- Bidirectional by default

Socket domains

Socket Domains

- The same API allows the creation of different sockets
- AF_UNIX
  - Communication between processes in the same machine
- AF_INET
  - Communication between processes in different machines
  - IPv4
- AF_INET6
  - Communication between processes in different machines
  - IPv6

Socket Domains

- Determines the nature of the communication (local/LAN/WAN)
- Determine the format of the addresses
  - AF_UNIX – a string
  - AF_INT – 4 bytes

Socket address

- Some operations require an address
  - Bind / connect
  - Sendto / recvfrom
- struct sockaddr *src_addr
  - Placeholder for various address classes
    - sockaddr_un - unix
    - sockaddr_in - IP
    - sockaddr_nl - netlink
    - sockaddr_atalk - appletalk

```
struct sockaddr {
                sa_family_t
sa_family;
                char
sa_data[14];
            }
```

Protocol

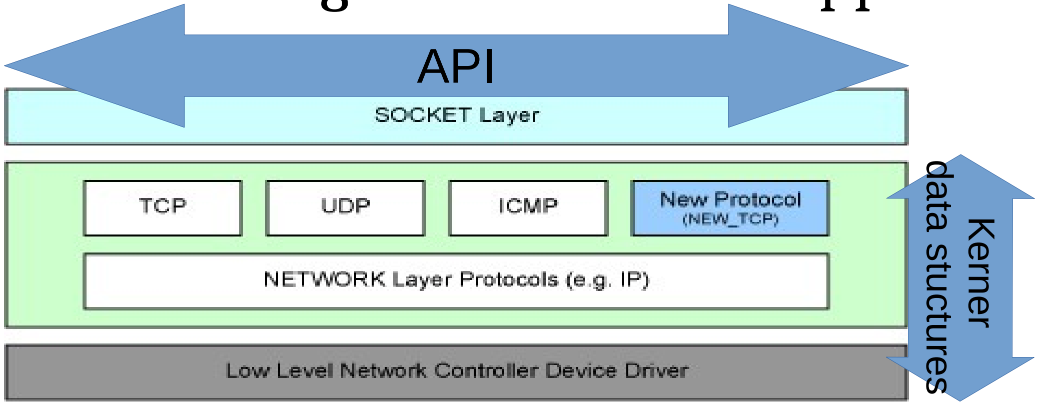
- The communication protocol
  - Depends on the socket Type and domain
  - Not all combinations are possible
  - Defines communication characteristics

		Domain		
		AF_UNIX	AF_INET	AF_UNSPEC
Type	SOCK_STREAM	**YES**	**TCP**	SPP
	SOCK_DGRAM	**YES**	**UDP**	IDP
	SOCK_RAW		**IP**	Sim
	SOCK_SEQPACKET	YES		SPP

Protocol

- Defines
  - Addressing
  - Delivery guarantees
  - Message Structuring

- Affects
  - Kernel data structures
  - API
  - Application


API
SOCKET Layer
TCP
UDP
ICMP
New Protocol
(NEW_TCP)
NETWORK Layer Protocols (e.g. IP)
Low Level Network Controller Device Driver
Kerner
data stuctures

Sockets functioning

Datagram sockets

- Client
  - Socket creation
  - Address assignment

    Optional
  - Message sending
    - Explicit address

- Server
  - Socket creation
  - Address assignment

    Required
  - Reception of message

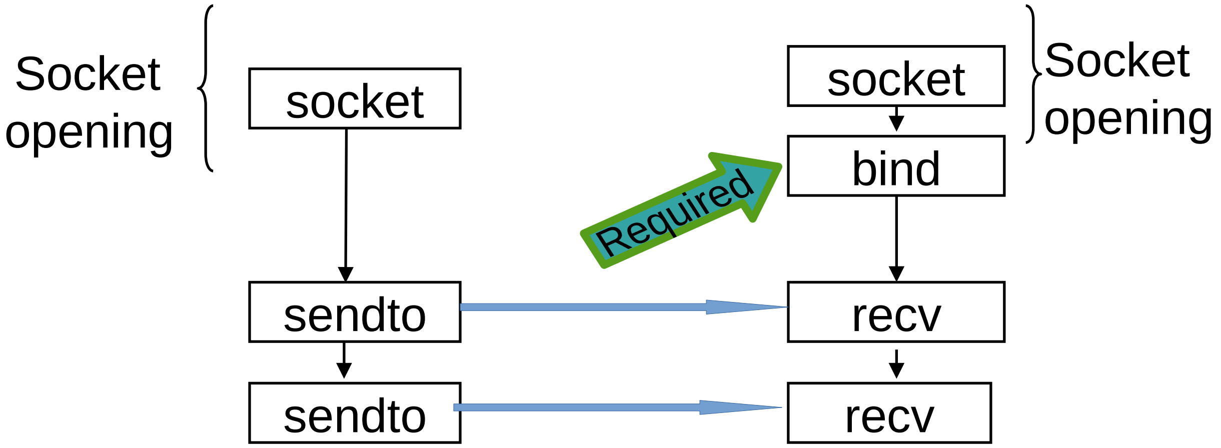
Datagram sockets

- Client
  - Socket creation
  - Address assignment
  - Message sending
    - Explicit address

Optional

- Server
  - Socket creation
  - Address assignment
  - Reception of message

Required



Datagram sockets

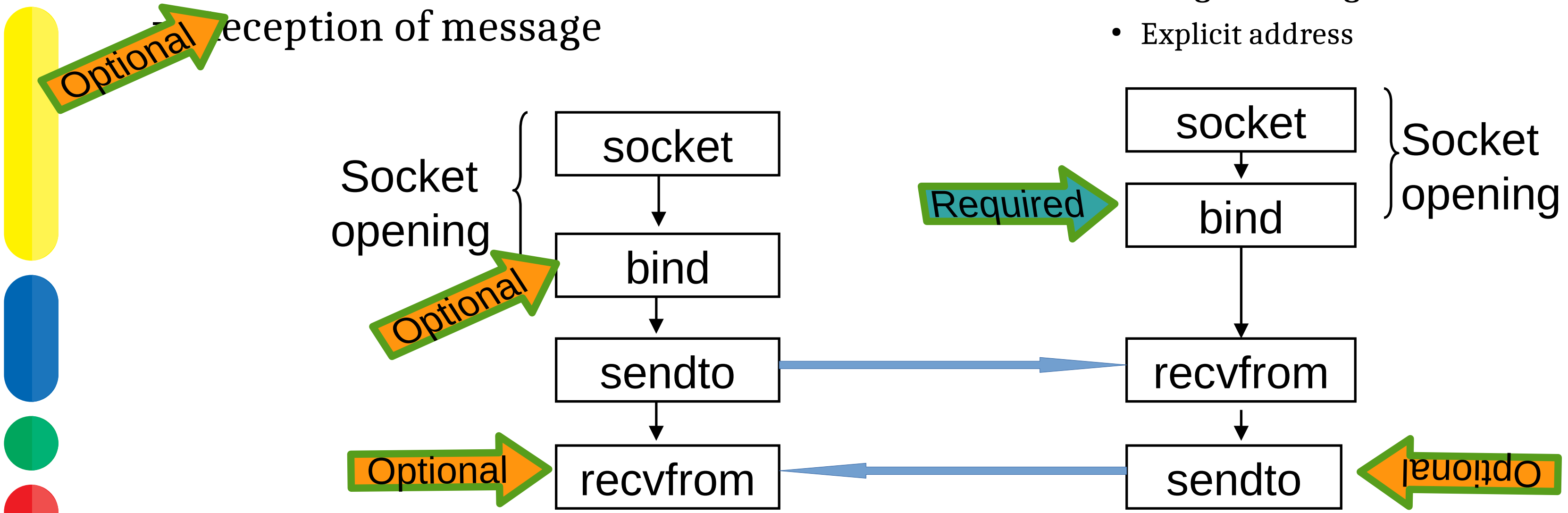
- Server
  - Socket creation
  - Optional Address assignment
  - Reception of message

- Client
  - Required Socket creation
  - Address assignment
  - Message sending
    - Explicit address

Datagram sockets

- Server
  - Socket creation
  - Address assignment
  - Reception of message

- Client
  - Socket creation
  - Address assignment
  - Message sending
    - Explicit address



Datagram sockets

- message oriented
- Without connection
  - Client just sends messages to server
  - Needs to address each message
- Without guarantees
  - Delivery
  - Order
  - integrity

- On the client
  - Create socket
  - Send messages
- On the server
  - Create socket
  - Assign address
  - Receive messages
  - One socket can receive messages from multiple clients
- Message transmission
  - sendto
  - Recv / recvfrom

Stream oriented sockets

Client

- Socket creation
- Connection to server
  - Explicit address
- Message sending
- Close

Server

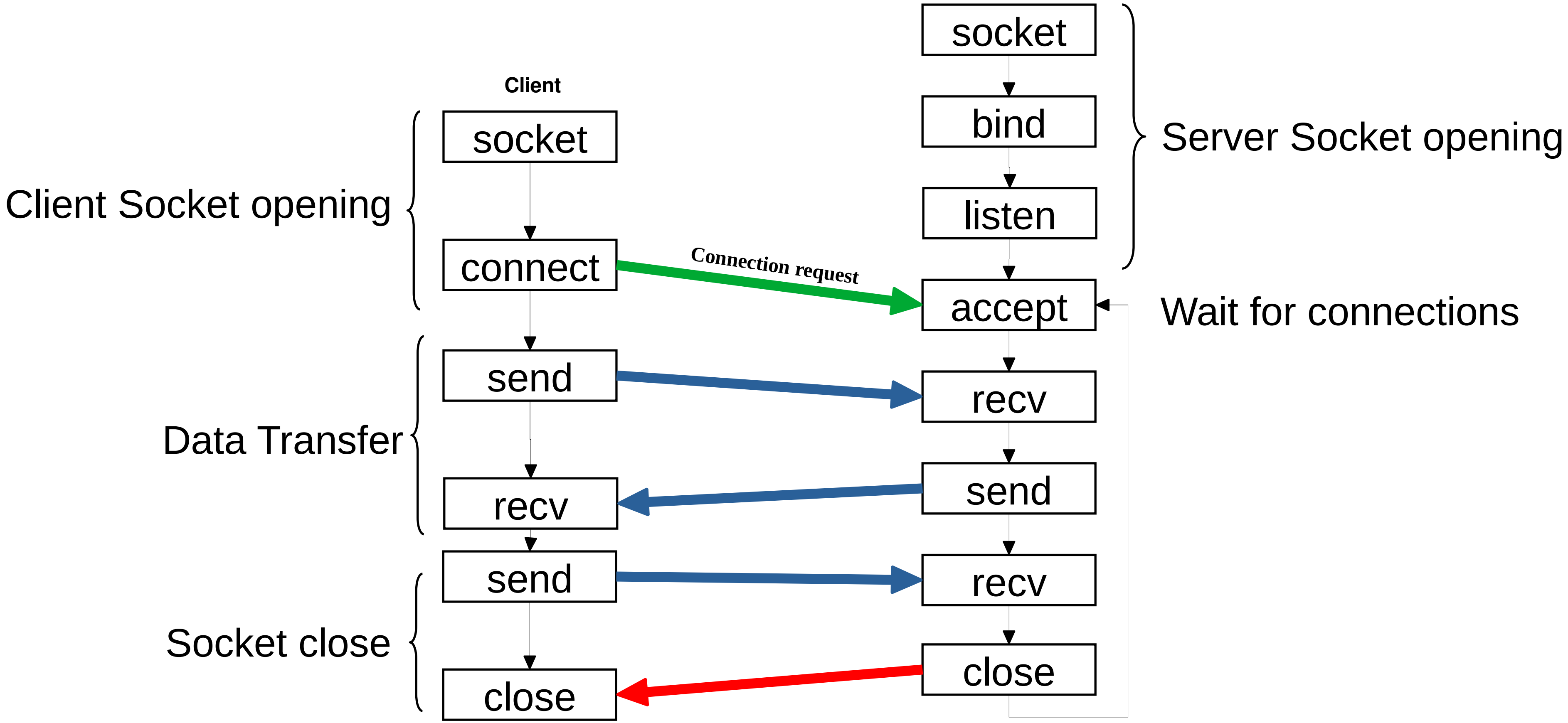
- Socket creation
- Address assignment
- Reception of connections
  - Connection acceptance
  - Reception of message
  - Close

Stream oriented sockets

- With connection
  - Client should connect to the server
  - Does not need to address each message
- With guarantees
  - Delivery
  - Order
  - integrity

- Connection establishing
  - Listen (server)
  - Connect (Client)
  - Accept (Serve
- On the server
  - A new socket is created
    - Dedicated to communication with the client
  - Original socket can receive more connections
- Message transition
  - read/recv
  - Write/send
  - Sendto/recvfrom
    - Not needed

Stream oriented sockets



Unix vs Internet sockets

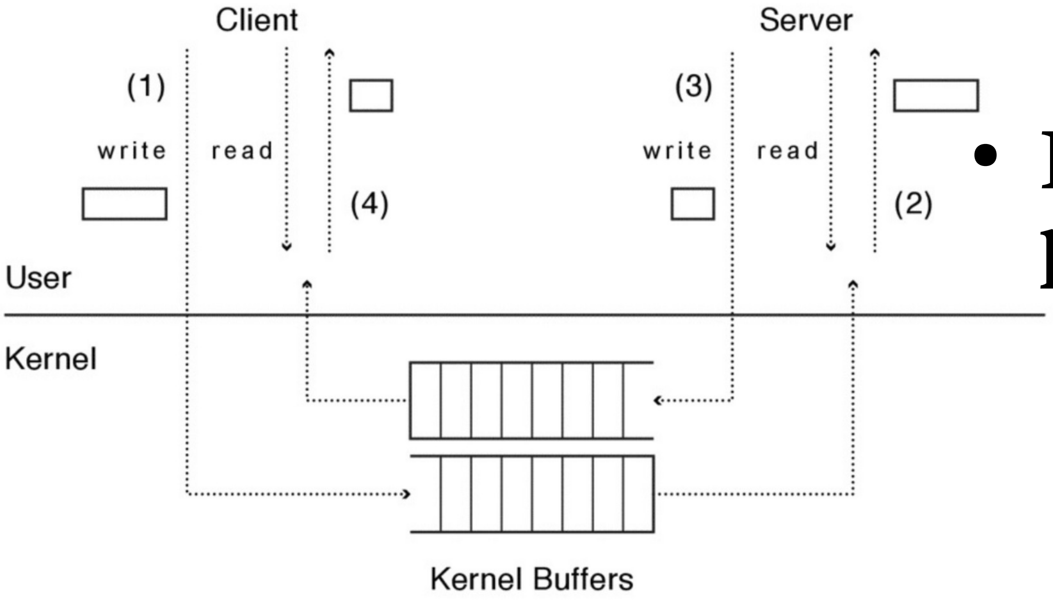
UNIX

- Addresses
  - String (file name)
- Data transmission
  - App → kernel → App
- Fault tolerance
  - All data delivered

Internet

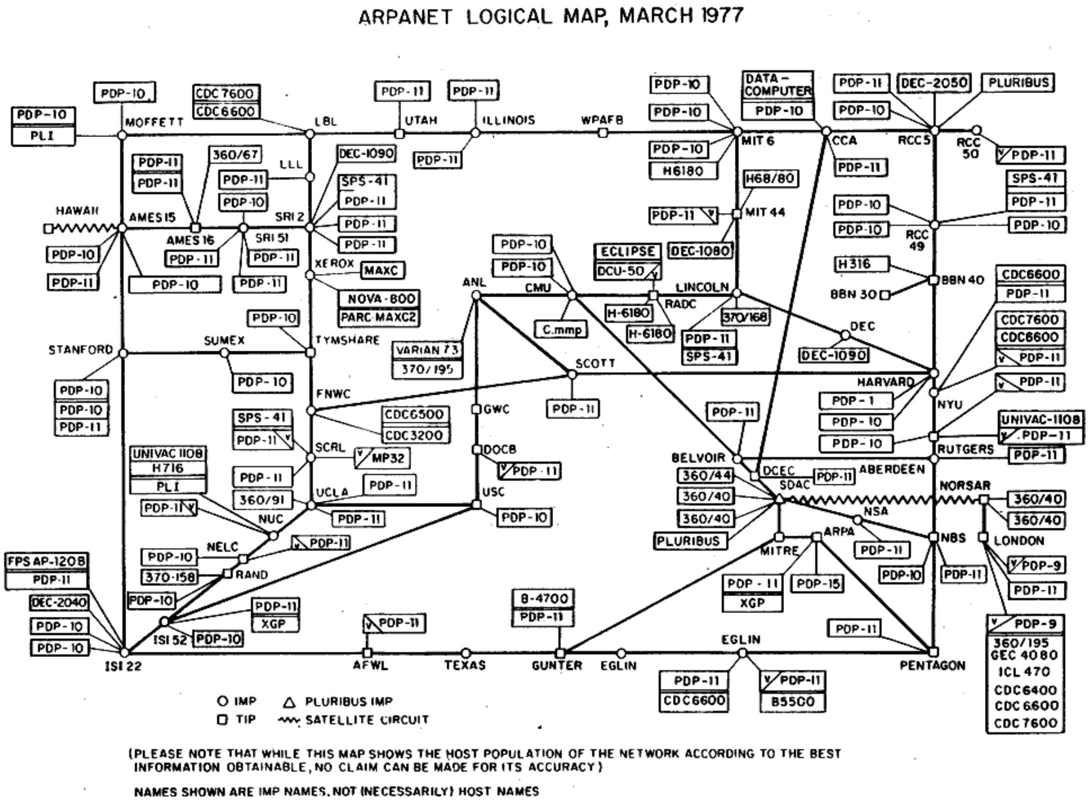
- Addresses
  - IP address + port
- Data transmission
  - App → kernel → network → kernel → App
- Fault tolerance
  - Depends on the protocol
- Data heterogeneity

Unix sockets data transmission



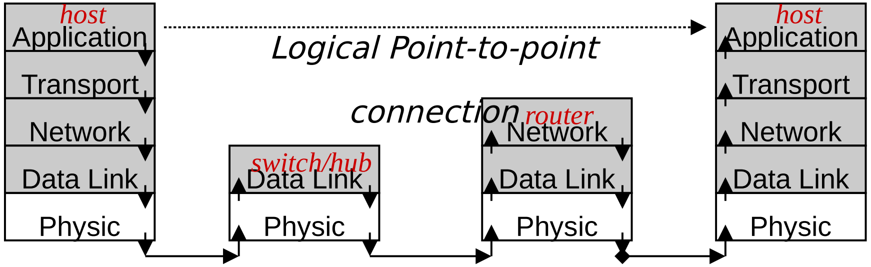
- Data is sent to kernel
  - send/sendto
- Data is stored in the kernel
  - Until other process read it

Unix sockets data transmission


ARPANET LOGICAL MAP, MARCH 1977

(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)

NAMES SHOWN ARE IMP NAMES, NOT (NECESSARILY) HOST NAMES



Unix sockets data transmission

- Data is sent to kernel
  - send/sendto
- Data is stored in the kernel
  - It can be forwarded to other computer
- Protocols guarantee that messages
  - Transverse the whole Internet
  - Reach the correct computer

Sockets fault tolerance

UNIX

- All data delivered
- Kernel stores all data until it is read

Internet

- Message oriented sockets
  - Messages can be lost
  - Message order is not guaranteed
  - Reduce message processing overhead
- Stream oriented message
  - Add data is transmitted and in the correct order
  - High communication overhead

Sockets addresses

UNIX sockets addresses

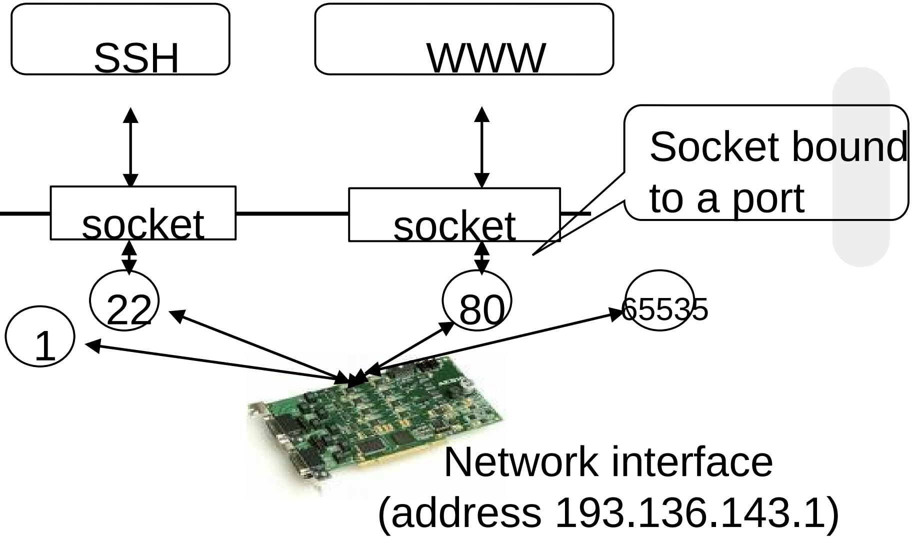
- Unix domain addresses are defined using
- **sockaddr_un** data type
- Definition of the domain
- **sun_family = AF_UNIX**
- Definition of the socket path
- **Strcpy(addr.sun_path, “/tmp/sock_1”)**

```
#include <sys/un.h>
struct sockaddd_un addr;
addr.sun_family = AF_UNI;
strcpy(addr.sun_path, “/tmp/sock_1”);
```

Internet sockets addresses

- Client needs to identify server
  - On a remote machine
  - From multiple services on such machine
- Internet domain addresses are defined using
  - Sockaddr_in data type
- Definition of the domain
  - sun_family = AF_INET

```
#include <netinet/in.h>
sockaddr_in local_addr;
i_addr.sin_port = htons( 22);
i_addr.sin_addr.s_addr = INADDR_ANY;
inet_aton("146.193.41.1", &i_addr.sin_addr)
```



Internet sockets addresses

- A service is identified by
  - Network address + Port
- The transmission/reception of data is made using a
  - Port - HW and operating System
  - Socket – App;lication
- A socket can be connected one of 64K ports.
- The first 1K ports (1-1023) are reserved to official services
  - specific services (listed in /etc/services)
  - 22 : SSH 53 : DNS 80 : WWW 115 : secure FTP 443 : secure WWW
- http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers
- Ports in between 49152-65535 should not be used by servers
  - Are assigned dynamically to client sockets

Review

Datagram vs stream

- Datagram
  - 1 socket multiple clients
  - Disconnect
    - Impossible
  - Messages limited size
  - No guarantees
    - ordering
    - Delivery
  - lightweight

- Stream
  - 1 socket 1 client
  - Disconnect
    - acknowledged
  - Streams of unlimited size
  - Guarantee
    - Ordering
    - delivery
  - heavy weight

Common Errors

- Common errors with socket programming:
  - Incorrect byte ordering
    - Not calling hton() e ntoh().
  - Disagree on data size and limits (fix or variable).
  - Non initialization of len (recvfrom, accept)
  - Locks
    - Application level protocol not well defined

Blocking calls

- Most socket API calls are blocking
  - When the process calls such function it gets blocked waiting for an event.
  - Accept - Waits for a connection to be received
  - Connect - Waits for the server to accept the connections
  - recv,recvfrom - Waits for data to be received
  - send,sendto - Waits for data to be transmited to a lower layer.
- In simple application blocking is good:
  - Adds synchronization
  - Limits resource usage (avoids active wait)

Blocking calls

- In complex applications blocking is a problem:
  - Multiple connections are impossible
  - Simultaneous send/receive are difficult
- There are several solutions
  - Multi-programming
    - Several processes or several threads
      - More complex programming
      - Synchronization required
  - Turn off blocking
    - More complex programming
      - Active wait
  - Use select
    - Wait on multiple descriptors
      - Serializes communication

Blocking calls

```
int select(int,fd_set *,fd_set *,fd_set *, struct timeval *);
```

- 1st parameter
  - Identifies the number of the highets descriptor + 1.
- 2nd parameter
  - array of reading descriptors (if set select verifies if such descriptors have information to be read)
- 3rd parameter
  - array of writing descriptors (if set select verifies if such descriptors can be writen to).
- 4th parameter
  - array of “exceptions” (if set select verifies if such descriptors has exception.
- 5th parameter defines the waiting interval (NULL to infinite wait).
- The function returns the number of affected descriptors Return -1 in case of error

Blocking calls

- descritors are refered in a bit array of type **struct fd_set** .
- Auxiliary funtions:
  - void FD_ZERO(fd_set *); /* crealr array */
  - void FD_CLR(int, fd_set *); /* set bit to 0 */
  - void FD_SET(int, fd_set *); /* set bit to 1 */
  - int FD_ISSET(int, fd_set *); /* return bit value */
- The select function activates the correct bits on the correct arrays, depending
  - on the input array
  - on the state of the descriptor

Multiple clients

- Select
- Fork
  - FD are Inherited
  - Read retrieve messages from same socket
  - Accept can be done in multiple processes
- Threads